453rd EWS Neural Network for System Identification

Caleb Vourazeris

Background

Inside Work

- Salisbury University Graduate | 4.0 GPA
  - SMART Scholar
  - Physics Bachelors
  - Summa Cum Laude
  - Excellence in engineering award
- University of Maryland Graduate | 3.864 GPA
  - Electrical Engineering Bachelors
- Intern at the 57th Intelligence Squadron
  - Combat Support Database Analyst
- Full time engineer at the 453rd Electronic Warfare Squadron
  - Initially IMOM (improved many on many) team
  - Lacked guidance, tasking, urgency, and engineering
  - Self taught software engineering
  - MATLAB courses, transitioned into python
  - Created a vision for AI/ML development at 453 EWS
  - Began building model for system identification

Outside Work

- Brazilian Jiu Jitsu
  - White Belt (Blue Belt Test 3/30/205)
  - Consistently trained for 1 year +
  - 3rd Place at Houston Fall International Open IBJJF Jiu Jitsu Championship
  - 3rd Place at Dallas Winter International Open IBJJF Jiu JItsu Championship 2025
- Guitar
  - Gig performance
  - Weekly lessons
- Poker
  - Playing since 2022
  - Play regularly for fun
  - Low stakes live poker
- Investing
  - Personally managed portfolio
  - Investing since 2019
- Weight Lifting
  - Lift regularly to stay in shape and socialize

Project Scope and Goal

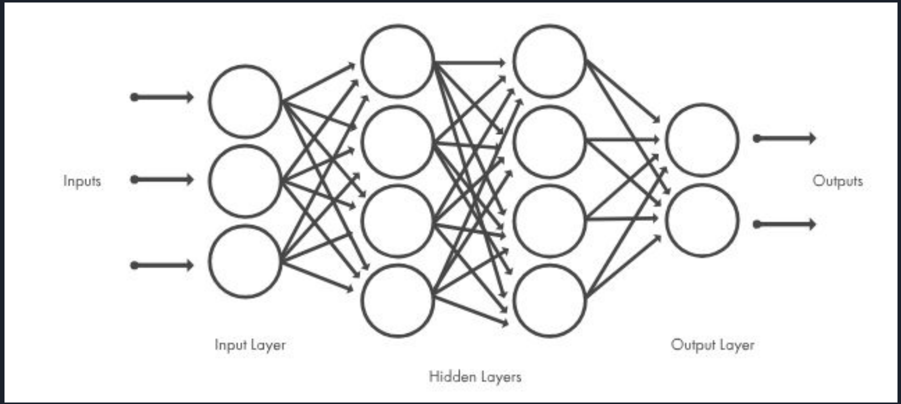
- Inception
  - Drew connection between MATLAB course datasets and 453EWS datasets
  - Analysts work was simply a classification problem
  - Decided to attempt to create a solution
- Problem
  - Automatically characterize signals based on parameters
  - Current Process
    - Rule based flagging system
    - Flags signals were manually inspected
- Issues with current process
  - Not working, a large majority of data not being processed
  - Rules based system is not robust
  - Manual inspection is not reliable (using geo data)
- Impact
  - Process <u>all</u> data that came through the pipeline
  - Automatically characterize signals
  - Robust solution

Responsibility

- Project Design Theory
  - Neural Networks
    - Classify handwritten numbers
    - 3 Blue 1 Brown Video Series: Neural Networks
- Project Design Implementation
  - Mock Code Written Using Iris Dataset
  - Code written using Pytorch, Numpy, Pandas etc
- Neural Network for System Identification
  - Project Summary
  - Results

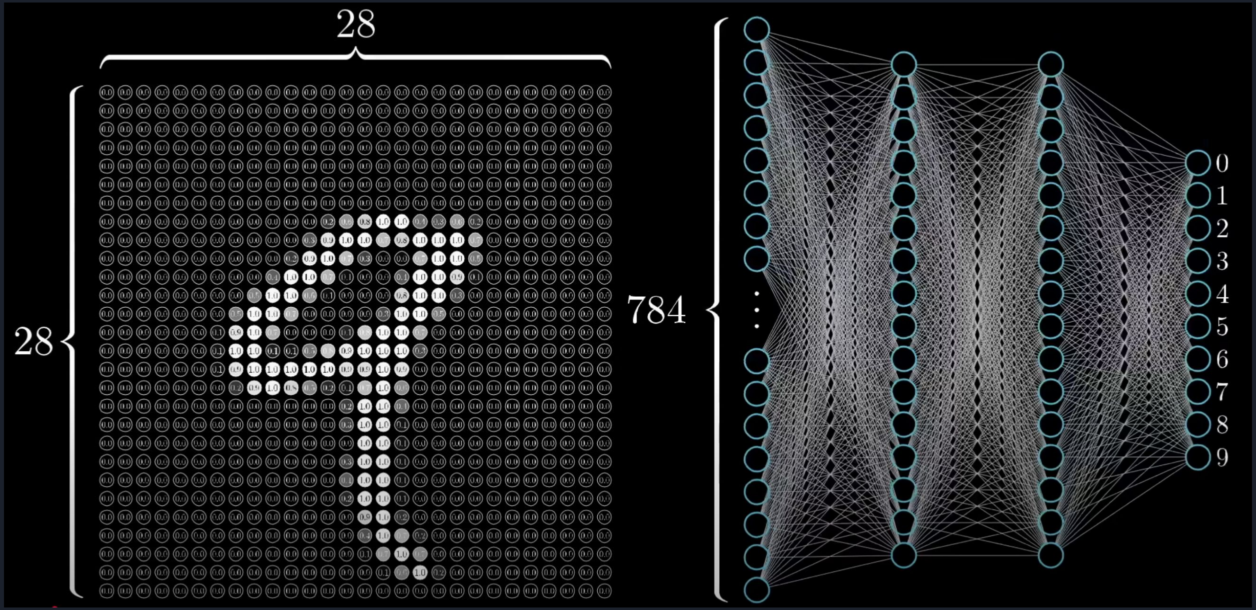
Project Design Theory

- Initial Activation Layer
- Neuron Weights
- Network Structure
- Cost Function
- Gradient Descent
- Backpropagation



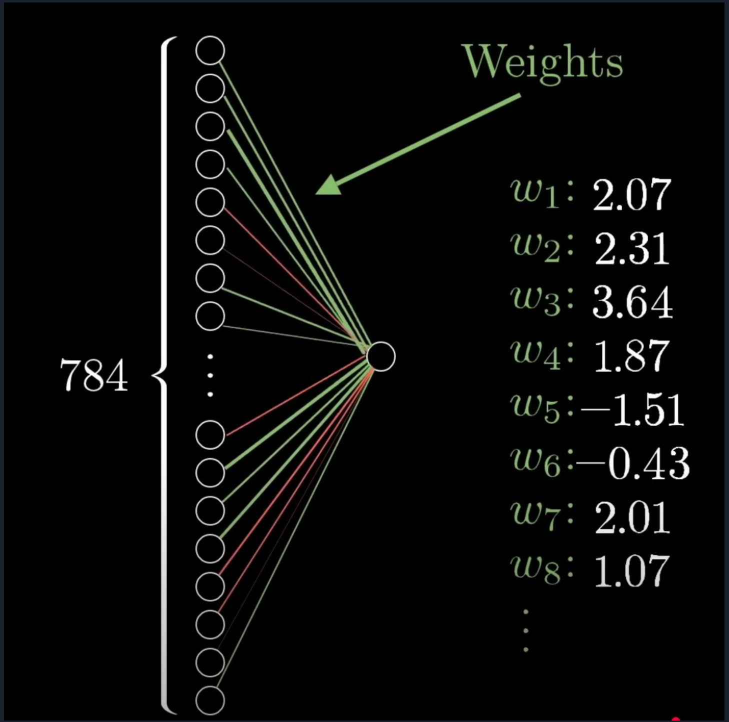
Project Design Theory: Initial activation layer

- Examples of hand drawn numbers
- Numbers are represented digitally by 28x28 drawings with grayscale values
- 28 x 28 = 784 makes up first layer of network



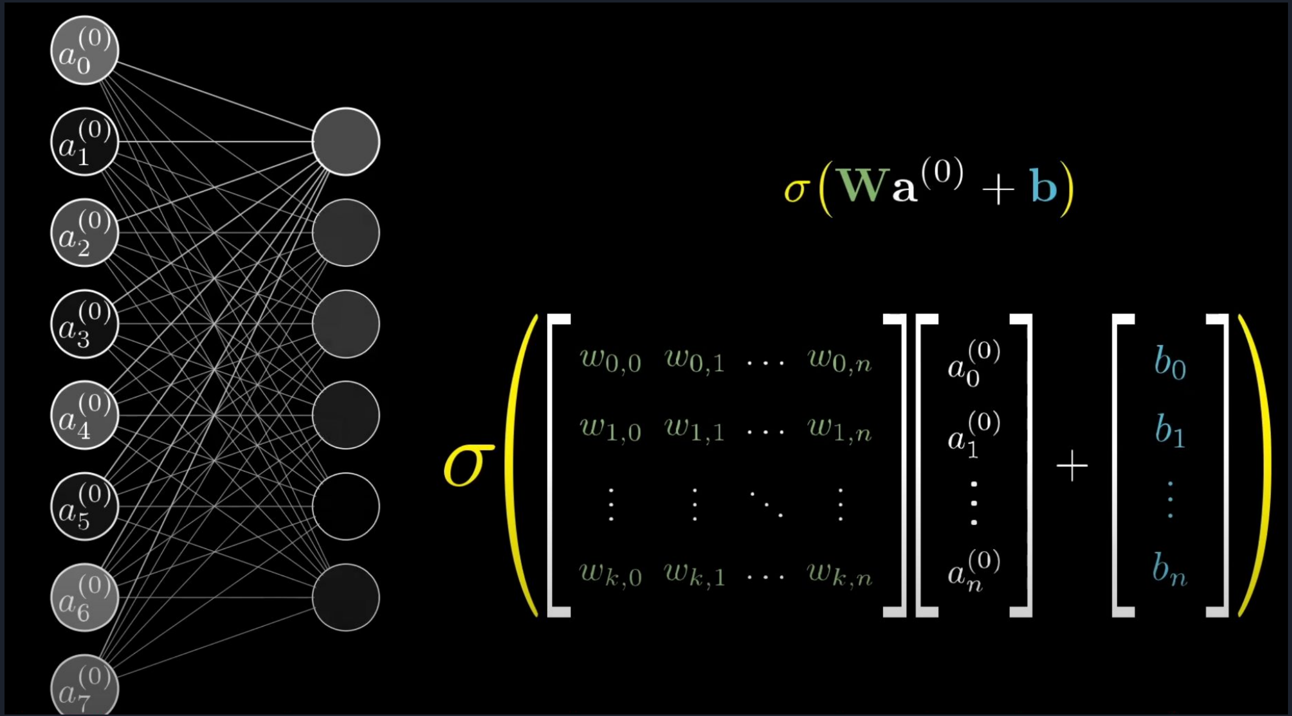
Project Design Theory: Neuron Weights

- Every neuron in each subsequent layer, is connected to every neuron in the previous layer
- These layer connections have weights
- In this example, the first hidden layer has 16 neurons, each having 784 weights



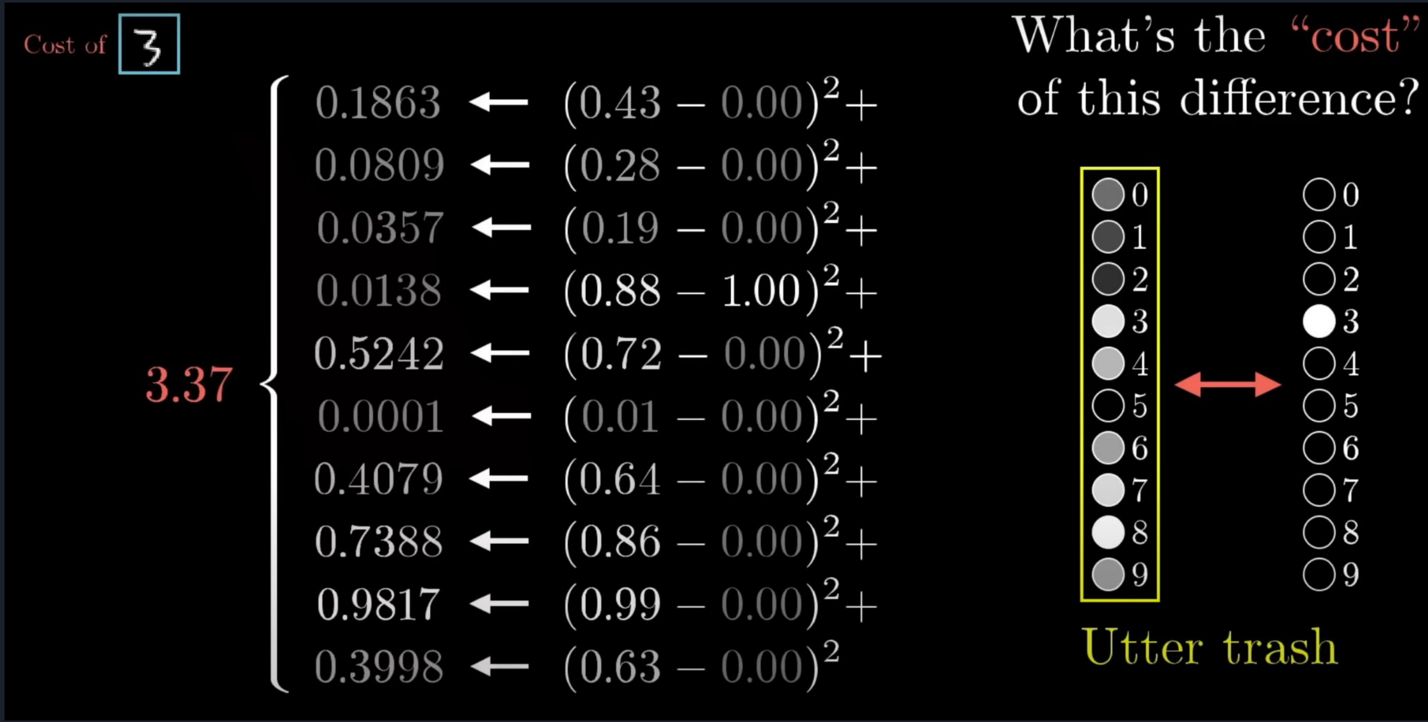
Project Design Theory: Summary of network structure

- Weights
  - 784 x 16 + 16 x 16 + 16 x 10 = 12,960
- Biases
  - 16 + 16 + 10 = 42
- Total number of parameters = 13,002
- Bias is how high weighted sum needs to be before the neuron becomes meaningfully active
- Sigmoid activation to squeeze numbers from 0 to 1



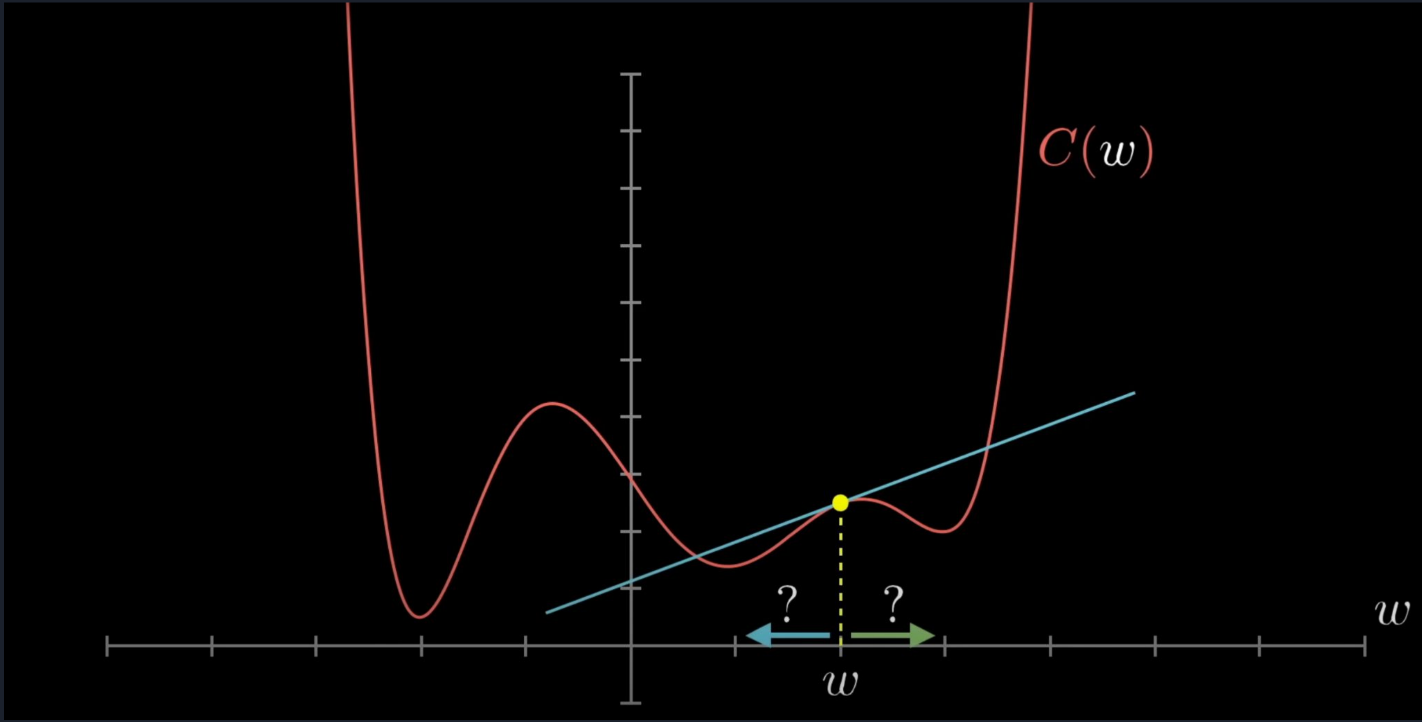
Project Design Theory: Cost Function

- Initial weights and biases are all completely random
- Throw initial training example through model
- Add up the squares of the differences between each of the models output activations and the desired value
- Consider average cost over all training examples



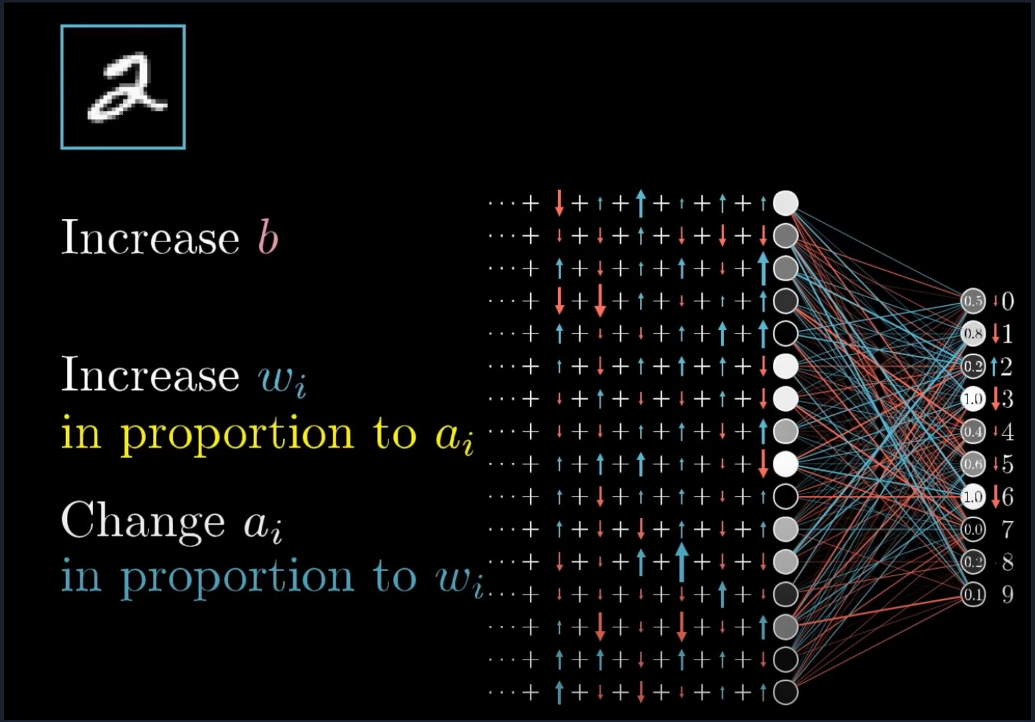
Project Design Theory: Gradient Descent

- Start at any random input
- Calculate the gradient to figure out which direction to make the cost function lower
- Take a tiny step in that direction
- Repeat the process until local minimum is found
- Finding a global minimum is too complicated



Project Design Theory: Back propagation

- Small adjustments need to be made for each classification
- Wrong classifications probabilities must decrease
- Correct probabilities need to increase
- Positive weights need to be increased
- Negative weights need to be decreased



Project Design Implementation

- Iris Dataset: Tabular Data
- Model Creation
- Custom Dataset Class
- Training Loop
- Model Ensemble

Project Design Implementation: Iris Dataset

- Tabular Data
- Similar Dataset to Signal Parameters
- Lacking Categorical Variables
- Very Small (only 150 examples)

	A	B	C	D	E
1	sepal.length	sepal.width	petal.length	petal.width	variety
2	5.1	3.5	1.4	0.2	Setosa
3	4.9	3	1.4	0.2	Setosa
4	4.7	3.2	1.3	0.2	Setosa
5	4.6	3.1	1.5	0.2	Setosa
6	5	3.6	1.4	0.2	Setosa
7	5.4	3.9	1.7	0.4	Setosa
8	4.6	3.4	1.4	0.3	Setosa
9	5	3.4	1.5	0.2	Setosa
10	4.4	2.9	1.4	0.2	Setosa
11	4.9	3.1	1.5	0.1	Setosa
12	5.4	3.7	1.5	0.2	Setosa
13	4.8	3.4	1.6	0.2	Setosa
14	4.8	3	1.4	0.1	Setosa
15	4.3	3	1.1	0.1	Setosa
16	5.8	4	1.2	0.2	Setosa
17	5.7	4.4	1.5	0.4	Setosa
18	5.4	3.9	1.3	0.4	Setosa
19	5.1	3.5	1.4	0.3	Setosa
20	5.7	3.8	1.7	0.3	Setosa
21	5.1	3.8	1.5	0.3	Setosa
22	5.4	3.4	1.7	0.2	Setosa
23	5.1	3.7	1.5	0.4	Setosa
24	4.6	3.6	1	0.2	Setosa
25	5.1	3.3	1.7	0.5	Setosa

Project Design Implementation: Model Creation

```
import torch.nn as nn
import torch.nn.functional as F
import torch

# Create a Model Class that inherits nn.Module
class Model(nn.Module):
  # Input layer (4 features of the flower) -->
  # Hidden Layer1 (number of neurons) -->
  # H2 (n) -->
  # output (3 classes of iris flowers)
  def __init__(self, in_features=4, h1=8, h2=9, out_features=3):
    super().__init__() # instantiate our nn.Module
    self.fc1 = nn.Linear(in_features, h1)
    self.fc2 = nn.Linear(h1, h2)
    self.out = nn.Linear(h2, out_features)

  def forward(self, x):
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.out(x)

    return x
```

Project Design Implementation: Custom Dataset

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
import torch


class IrisDataset(Dataset):
    def __init__(self, dataframe):
        dataframe = self.label_change(dataframe)
        self.features = dataframe.drop('variety', axis=1)
        self.labels = dataframe['variety']
        self.tensor_transform()

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        features = self.features[idx]
        labels = self.labels[idx]
        return features, labels

    def label_change(self, dataframe):
        dataframe['variety'] = dataframe['variety'].replace('Setosa', 0)
        dataframe['variety'] = dataframe['variety'].replace('Versicolor', 1)
        dataframe['variety'] = dataframe['variety'].replace('Virginica', 2)
        return dataframe

    def tensor_transform(self):
        self.features = torch.FloatTensor(self.features.values)
        self.labels = torch.LongTensor(self.labels.values)
```

Project Design Implementation: Training Loop

```python
def train_model(model, train_dataloader, validate_dataloader, criterion, optimizer, num_epochs):
    # Train our model!
    model.train()

    # Epochs? (one run thru all the training data in our network)
    train_epoch_loss = []
    validate_epoch_loss = []

    for epoch in range(num_epochs):

        batch_losses = []

        for data in train_dataloader:

            features, labels = data

            # Go forward and get a prediction
            y_pred = model(features) # Get predicted results

            # Measure the loss/error, gonna be high at first
            loss = criterion(y_pred, labels) # predicted values vs the y_train

            # Do some back propagation: take the error rate of forward propagation and feed it back
            # thru the network to fine tune the weights
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            batch_losses.append(loss.item())

        # Keep Track of our losses
        train_epoch_loss.append(sum(batch_losses)/len(batch_losses))

        val_loss = validation_loop.validate_model(model, validate_dataloader, criterion)

        validate_epoch_loss.append(val_loss.item())


        print(f'Epoch: {epoch} Training Loss: {train_epoch_loss[epoch]} Validation Loss: {validate_epoch_loss[epoch]}')

    return model, train_epoch_loss, validate_epoch_loss
```

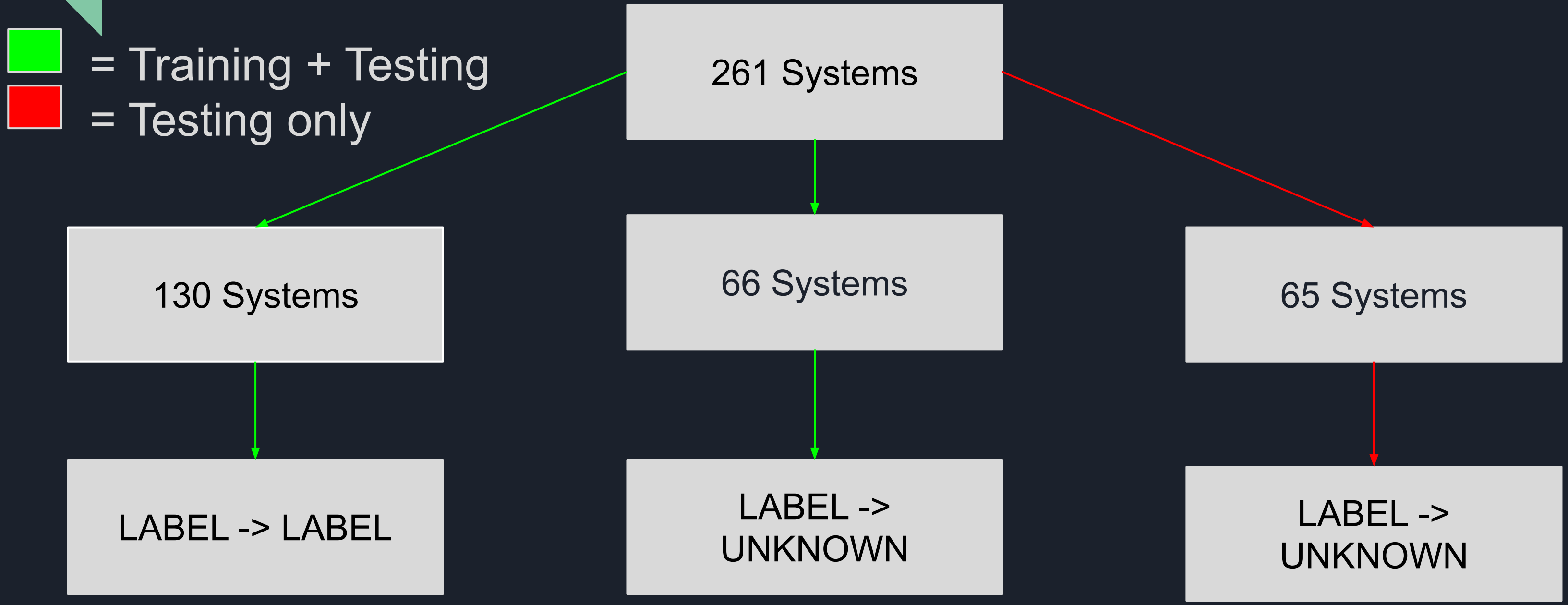
Project Design Implementation: Model Ensemble

```python
class ModelEnsemble():

  def __init__(self, models_list, test_dataloader):
    self.models_list = models_list
    self.test_dataloader = test_dataloader

  def run_inference(self):
    outputs_list = []
    labels_list = []
    predictions_list = []
    for model in self.models_list:
      _, _, predictions, labels, outputs = evaluate_model.test_model(model, self.test_dataloader)
      outputs_list.append(outputs.flatten(start_dim=0, end_dim=1))
      labels_list.append(labels.flatten())
      predictions_list.append(predictions.flatten())

    self.model_outputs = outputs_list
    self.labels = labels_list
    self.predictions = predictions_list

  def majority_vote(self):
    model_output_sum = sum(self.model_outputs)
    ensemble_output = torch.argmax(model_output_sum, dim=1)
    self.ensemble_output = ensemble_output
```

Neural Network For system Identification: Summary

- Queried Database for our Intercepts
  - Queried a minimum of 5,000 and a maximum of 10,000 intercepts
  - Signals had varying amounts of intercepts in the database
- Every intercept had an original system label
  - Total of 261 systems
  - Kept original system label for first half (130 systems)
  - Second half changed original system labels to ‘UNKNOWN’ (a total of 131 systems)
  - Used <u>first</u> half (65 unique systems) of the ‘UNKNOWN’ data to train the algorithm
    - Resulted in a category that had a wide variety of signals it has seen
    - The idea was to any signal the model hadn’t been trained on would be dumped here
  - We used the second half (66 unique systems) of the ’UNKNOWN’ data to test the algorithm
    - These were systems the model has never seen examples of
    - These were supposed to represent new systems in the electromagnetic environment
    - The hope was these would be dumped into the ‘UNKNOWN’ category

Diagram



System had 131 outputs: 130 unique system labels, then 1 for ‘UNKNOWN’

Neural Network For system Identification: Results

- Final result metrics
  - Overall classification testing accuracy was 95%
  - On the training data that had the original system label (130 systems)
    - Most categories had 97-100% accuracy
    - Model performed very well on these
  - On the training portion of ‘UNKNOWN’ data
    - Model performed well 97%+ accuracy
  - On the training data the model had never seen (66 systems)
    - Supposed to be classified as unknown
    - Using the model ensemble, achieved roughly 76% accuracy
    - Means some of these systems were misclassified as one of the original system labels

Challenges

- Acquiring Development Software
  - Solved by investing in relationship with software team
  - Got approvals quickly and efficiently
  - Frequently checked in on tickets
- Locating Hardware to train algorithm
  - Searched multiple avenues for potential solutions
  - Poor: Acquire hardware at squadron
  - Best: Acquire account for free HPC hosted by Air Force Research Lab
  - Found a POC, reached out, got in touch with accounts team
- Dealing with Unknown Signals
  - Model was trained on subset of known systems
  - Needed a solution for systems the model was not trained on
  - Created unknown category for all systems the model wasn't trained on
  - Poor: Tricky problem, started with tweaking the probability threshold for classification
  - Robust: A better solution was to use a model ensemble, with a default of 'UNKNOWN'

Lessons Learned

- Take ownership and responsibility
  - Best way to grow technically
  - Increased motivation and performance
- Leverage other people's skills
  - Humility is a great tool, know what you don't know
  - Find smart people, ask them for help
- Follow up consistently with requests
  - People have their own agendas
  - Make sure they don't forget about yours
- Be vocal about what you're working on
- Jump into learning new things
  - Headfirst is the best way to go
  - Learned Python, Linux, SQL, MATLAB
- Briefing leadership
  - Do not bog them down on technicals
  - Keep briefings very short, 3 slides if possible
  - Clearly define WHY
  - Focus on impact, result, and metrics
  - Give them timelines and future plans

Takeaways: Could recreate project in 1/3rd of the time (3-4 months). In a more technically advanced environment, I'd lean more heavily on other professional developers

Final Impact of Project

As of my departure, project has not been deployed. Still several positive takeaways

- Development environment on high side network
  - Python, Git, VScode approvals
  - On government highside, approvals take weeks sometimes months
- Multiple resources were located
  - High Performance Compute Cluster
  - NSA python package repository
- Machine Learning Foundation
  - Github repository
  - Initial code and project created for further development
- Additional Engineers Setup for success
  - After I left, 2 engineers had HPC accounts, and development software installed
  - Those engineers are responsible for getting projected deployed

Why SpaceX and Starshield?

- Similar Mission
  - Satellites and data
  - Government focused
- Technical Expertise
  - Previous employment lacked technical experience
  - SpaceX has some of the best engineers in the world
  - Hoping to learn from peers
- Ambition, purpose, and drive
  - Previous employment lacked ambitious, driven, lifelong learners
  - Takes their missions seriously, there’s a lot on the line, everyone’s work matters
  - Culture of excellence and merit